Apple Assembly Line

$1.80

Volume 5 -- Issue 4 January, 1985

In This Issue...

Note about Apple Manuals

We have mentioned before how hard it is to find the Apple technical manuals, but it looks like there is now hope. We read somewhere this week that Apple has arranged for Addison-Wesley to distribute the manuals. If this really comes to pass, we will probably be able to get them for you like any bookstore. Here's hoping!

New Version of 6800/6801/6301 Cross Assembler

We have started the long process of upgrading the various S-C Cross Assemblers, and the first one is now available. Owners of Version 1.0 of the 6800/6801/6301 Cross Assembler and of the Version 2.0 of the S-C Macro Assembler can upgrade to Version 2.0 of the Cross Assembler for $20.

If you have not already upgraded to Version 2.0 of the S-C Macro Assembler (for the 6502 et al), you need to do that first or at the same time. If you already have 6502 Version 2.0, but don't have the older version of the 6800 product, you can go directly there for only $50.

6800 XASM Version 2.0 adds 80-column support (for //e, //c, Videx, and STB-80 users), five new directives, and all the other bells and whistles of our 2.0 products.

New disk price!!

Due to incredible competition, floppy disk prices are falling almost as fast as if they were semiconductors! Check our ad on page three for the current low price.

18-Digit Arithmetic, Part 9.....Bob Sander-Cederlof

Nearing the home stretch, this month I will cover the DP18 PRINT statement. I believe that only leaves INPUT for next month.

Normal Applesoft PRINT has a wide variety of options. PRINT may appear all by itself to print a carriage return, or with one or more expressions. The expressions may be separated by commas or semicolons: both are used to separate the expressions for syntax purposes, but commas also cause a form of tabbing. A final comma or semicolon may be used to suppress the normal carriage return at the end of the printed line. All numeric values are printed in an unformatted style.

We wanted to have additional formatting capabilities in DP18 PRINT. Many users of Applesoft have tried to write money handling programs, agonizing over the contortions necessary to make pretty reports. BASIC on many other micros comes with PRINT USING, which includes a string describing the exact format to use for print a list of items. Applesoft doesn't have PRINT USING (we have graphics instead, and all in a 10K interpreter). DP18 does.

DP18 doesn't have everything though. Here are some things we left out. Commas may be used to separate items in a DP18 PRINT statement, but no tabbing happens. Instead, commas cause carriage returns. DP18 values are so long that comma tabbing seemed useless. You cannot fit two fully extended unformatted values in one 40-column line. Maybe you could say we do tab, all the way to the next line. Anyway, this gives us a useful NEW feature: the ability for one PRINT statement to print on more than one line.

DP18 PRINT can only print DP18 expressions. Normal Applesoft real or integer expressions can be printed by normal Applesoft PRINT, or by converting them to DP18 values using VAL and STR$. Applesoft string expressions can be printed using a DP18 "picture", but not in the simple manner you are used to in normal Applesoft PRINT.

DP18 in its present form supports three different kinds of items in a PRINT statement: DP18 expressions, #WD items, and $PIC items.

The first kind is the easiest to use, and will remind you a lot of Applesoft. Since all you tell DP18 is the expression, it makes up its own mind about the format to use. We call this "unformatted", because it hard to predict how it will look once it is printed. If the absolute value of the number to be printed is within the range from .01 to 999,999,999,999,999,999 (18 digits) it will print as a normal number, with no leading or trailing blanks and no trailing zeroes. If outside that range, it will be printed with an E exponent. Doesn't this remind you of Applesoft? Here are some examples using numbers (bear in mind they could be long complicated DP18 expressions):

S-C Macro Assembler Version 1.0.....$80
S-C Macro Assembler Version 2.0.....$100
Version 2.0 Update.....$20
Source Code for Version 1.1 (on two disk sides).....$100
Full Screen Editor for S-C Macro (with complete source code).....$49
S-C Cross Reference Utility (without source code).....$20
S-C Cross Reference Utility (with complete source code).....$50
DISASM Dis-Assembler (RAK-Ware).....$30
Source Code for DISASM.....additional $30

S-C Word Processor (with complete source code).....$50
Double Precision Floating Point for Applesoft (with source code).....$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk.....$15

(All source code is formatted for S-C Macro Assembler Version 1.1. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.....each $15
Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.

QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983	QD#11: Apr-Jun 1983	QD#12: Jul-Sep 1983
QD#13: Oct-Dec 1983	QD#14: Jan-Mar 1984	QD#15: Apr-Jun 1984
QD#16: Jul-Sep 1984	QD#17: Oct-Dec 1984	

AWIIe Toolkit (Don Lancaster, Synergetics).....$39
Visible Computer: 6502 (Software Masters).....(reg. $50) $45
ES-CAPE: Extended S-C Applesoft Program Editor.....$60
"Bag of Tricks", Worth & Lechner, with diskette.....($39.95) $36

Blank Diskettes (Verbatim)..... package of 20 for $35
(Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each.....10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).....40 cents each or $25 per 100
These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers..... 6 cents each
quikLoader EPROM System (SCRG).....($179) $170
D MAnual Controller (SCRG).....($90) $85
Switch-a-Slot (SCRG).....($190) $175
Extend-a-Slot (SCRG).....($35) $32
PROmGRAMER (SCRG).....($149.50) $140

Books, Books, Books.....compare our discount prices!

"Inside the Apple //e", Little.....($19.95) $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.($19.95) $18
"Apple][Circuit Description", Gayler.....($22.95) $21
"Understanding the Apple II", Sather.....($22.95) $21
"Enhancing Your Apple II, vol. 1", Lancaster.....($15.95) $15
Second edition, with //e information.
"Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95) $20
"Incredible Secret Money Machine", Lancaster.....($7.95) $7
"Beneath Apple DOS", Worth & Lechner.....($19.95) $18
"Beneath Apple ProDOS", Worth & Lechner.....($19.95) $18
"What's Where in the Apple", Second Edition.....($19.95) $19
"6502 Assembly Language Programming", Leventhal.....($18.95) $18
"6502 Subroutines", Leventhal.....($18.95) $18
"Real Time Programming -- Neglected Topics", Foster.....($9.95) $9
"Microcomputer Graphics", Myers.....($12.95) $12

Add $1.50 per book for US shipping. Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We accept Master Card, VISA and American Express ***

```
]&DP:PRINT 1,2,3;4;5
1
2
345
]&DP:PRINT .009,.01,999999999999999999
9E-3
.01
999999999999999999
]&DP:PRINT 1000000000000000000
1E+18
]
```

If a PRINT list item begins with the character "#", it is a #WD formatted item. Three things follow the "#" character, separated by commas: a field width, the number of fractional digits, and a DP18 expression. (If you have ever used Fortran, this is going to remind you of the "Fw.d" format.)

```
&DP:PRINT #w,d,value
```

The w and d parameters are Applesoft expressions (or simple constants), and the value is a DP18 expression. The value will be printed right-justified in a field w-characters wide, with d decimal places after the decimal point. Leading blanks will be printed if there is room for any. If the number will not fit in w characters, w asterisks will be printed instead to show you there was an overflow problem. Values are rounded to the required number of decimal places, not just truncated. Here are some examples:

```
]&DP:PRINT #8,3,2.04;#8,3,5,#10,5,3.14159,#3,1,99
   2.040    5.000
   3.14159
***
]&DP:PRINT #8,4,3.14159,#7,3,3.14159,#6,2,3.14159
  3.1416
  3.142
  3.14

100 FOR I=0TO5
110 PRINT I;:&DP:PRINT #10-I,5-I,3.1415926
120 NEXT
]RUN
0    3.14159
1    3.1416
2    3.142
3    3.14
4    3.1
5    3.
```

The third type of PRINT item begins with a dollar sign. A string constant, variable, or expression follows the dollar sign. If the picture specifies fields for DP18 or string values to be printed in, then the list of values must follow the picture, all separated by commas.

```
&DP:PRINT $ picture
&DP:PRINT $ picture,list
```

The "picture" is any Applesoft string expression; it is used as the template for formatting the expressions in the optional list. The list may have any number of expressions separated by commas as long as they correspond with the picture. You may even have no expressions at all, which is why I say the list is optional.

The picture consists of a string of characters. There are four basic types of characters used in pictures: commands, literals, numbers, and field descriptions. These are described below.

Any number in the picture makes up a repeat count. The repeat count specifies how many times to repeat the following command or field-description character. If a command is not preceded by a repeat count, a 1 is assumed. Repeat counts may range anywhere from 1 to 255.

The commands which may be included in pictures give you control over the screen and cursor. Some of the commands allow a repeat count to be specified. In the following descriptions, "n" refers to the optional repeat count. If no repeat count is used, n=1.

/ -- Prints n carriage returns.

X -- Prints n spaces.

> -- Clear to from the cursor to the end of line. If the next picture character is also ">", clear from the cursor to the end of screen.

V -- Performs VTAB n, where n must be from 1 to 24.

H -- Performs HTAB n. [As implemented now, this is probably not compatible with your printer or 80-column cards.]

Literals are defined in strings using the apostrophe. Any text you want to print from inside the picture may be included between apostrophes. If you want to include an apostrophe inside a literal, put two apostrophes in a row. If you put a repeat count before the literal, it will be printed n times.

Now here are some examples using repeat counts, commands, and literals.

```
&DP:PRINT $ "VH>>"      (moves the cursor to the top left
                         corner, and clears the screen.)
```

```
]&DP:PRINT $ "'SEPARATE'/'LINES'"
SEPARATE
LINES
]&DP:PRINT $ "4V10H3'BANG! '"
starting at line 4 column 10 prints:
          BANG! BANG! BANG!
```

There are two kind of field descriptions: one for telling DP18 how to print numbers and the other for telling how to print strings. Since string descriptors are easier, let's start with them.

A string field descriptor tells DP18 how to print the value of an Applesoft string. There are three different characters used, which tell DP18 whether to left-justify, right-justify, or center the value of the string within the field. Since we are building a "picture", the width of the field is shown by using multiples of the controlling character. The three different controlling characters are:

A -- Print the string left justified in the field.

R -- Print the string right justified in the field.

C -- Print the string centered in the field.

The data to be printed comes from the list of data items which follows the picture. Here are some examples using string descriptors:

```
]A$="ABC"
]P$="AAAAAAA'-'RRRRRRR'-'CCCCCCC'-'"
]&DP:PRINT $ P$,A$,A$,A$
ABC     -     ABC-  ABC  -
aaaaaaa.rrrrrrr.ccccccc.

]PRINT $"RRRRR X 5A 'HI' 6C 'BYE'","AB","ABC","XY"
   AB ABC  HI   XY   BYE
rrrrrxaaaaa..cccccc...
```

If you mix the A, C, and R control letters in one string field descriptor, the controlling letter will be the last one in the field. If you want to have two fields adjacent to each other, you can separate the descriptors with a space. The space will not become part of the printed output. If a string value is too long to fit in a field, the field will be filled with asterisks instead of the actual data. When you see asterisks where you expected data, the data was too long.

```
]&DP:PRINT$"AAA AAA AAA","AN","EGG","ROLLS"
AN EGG***
```

Numeric field descriptors are made up of the characters listed below. The number to be printed is taken from the expression list. The expression corresponding to a numeric field descriptor MUST be a DP18 expression. If it is not a DP18 numeric expression, an error will result. If the number is too

large for the field, asterisks will be printed. The number is rounded to the number of decimal places you specify in the descriptor before printing. Trailing zeroes after the decimal point are printed if necessary to fill up the field.

+ -- Reserves a place for the sign of the number. the sign will be printed in this position even if the number is positive. The sign may be placed anywhere within or at either end of the number.

- -- Also reserves a place for the sign, but the sign will only be printed if it is negative. If the number is positive, the fill character is printed instead.

(If neither + nor - is present in the field descriptor, the sign is printed only if the number is negative. It is printed just to the left of the first significant digit of the number. If you used zero or star fill, this looks ridiculous; therefore be sure to specify the sign position when you use zero or star fill.)

-- Reserves a place for a digit, and selects space fill. Unused digit positions to the left of the most significant digit will be filled with spaces.

* -- Reserves a place for a digit, and selects star fill. Unused digit positions to the left of the most significant digit will be filled with stars.

Z -- Reserves a place for a digit, and selects zero fill. Unused digit positions to the left of the most significant digit will be filled with zeroes.

. -- Reserves a position for the decimal point. The number will be lined up with the decimal point. If no decimal point is present in the picture, none is printed. Don't try to put more than one decimal point in one descriptor.

, -- Puts a comma in the number. If the comma would precede all the non-blank characters printed in the field, the comma will not be printed.

If a mixture of #, *, and Z characters are used in field descriptor, the field will be controlled by the last one.

```
]PRINT$ "'THE ANSWER IS '###,###.##",53156.6378
THE ANSWER IS  53,156.64

]PRINT$ "####.##+/####.##-/####.##+/####.##-",
12,12.3,-12.34,-12.345
```

```
12.00+
12.30
12.34-
12.35-

]PRINT$ "5Z.3Z",125.65
00125.650
```

The listing of the DP18 PRINT code follows. There are references to five subroutines printed in previous issues of AAL in lines 1220-1260. The subroutines INPUT.NUM and INPUT.STR which are also referenced will not be printed until next month. Ah, anticipation...!

When the &DP processor encounters a PRINT token, it jumps to DP.PRINT at line 1690. I like simple code, so you can see for yourself that DP.PRINT is only three lines long. All the work is done by PRINT.END (lines 2100-2420) and the routines it calls.

PRINT.END checks for ";" and "," separators between PRINT groups, and branches to the processors for each of the three types of PRINT groups. Lines 2110-2130 check whether we are at the end of the PRINT statement. If so, AS.CROUT prints a carriage return and we leave. If not at the end, a semicolon takes us down to line 2400. There we again check for the end, because a semicolon on the end of the PRINT statement means to omit the final carriage return. A comma takes us to line 2380 where we force-print a carriage return (DP18's kind of tabbing, remember).

Lines 2180-2210 check for the three possible types of PRINT groups: "$" means print with a picture, "#" means print with a w.d format, and anything else means unformatted printing. The #w.d type is handled right here in lines 2230-2360.

Lines 2230-2250, with the help of some code in the Applesoft ROMs, read the next characters from the PRINT statement, calculate whatever expression they represent, and save the result for the field width "w". Lines 2260-2300 do the same for "d". Line 2310 evaluates the DP18 expression for the data value to be printed. Lines 2320-2350 call on the FORMAT.PRINT subroutine discussed some months ago in AAL. After printing, we go back to the top of PRINT.END to allow another PRINT group.

Unformatted printing is handled in lines 1740-2080. Line 1750 evaluates the DP18 expression to be printed. Lines 1760-1800 decide whether to use normal or exponential format, depending on position of the decimal point. The exponential format is handled by QUICK.PRINT and the normal format by FOUT, both printed in an earlier installment. We call FOUT with a format of 40 characters wide and 19 places after the decimal point. Then we print only the significant digits of the resulting string. All leading blanks and trailing zeroes are omitted. If the last character is a trailing decimal point, it too is omitted.

Printing with a picture starts at line 2440. The picture processing code is also used by DP18's INPUT$ statement, and a simple flag is used to tell who called. PRINT sets the INPUT.FLAG = 1, INPUT sets it = 0. INPUT$ and PRINT$ join at line 2470.

The first step in picture processing is to make a working copy of the picture in DP18's PICTURE.BUF. Lines 2490-2510 evaluate the string expression which is the picture. Lines 2520-2650 copy the result into PICTURE.BUF, and place a terminating $00 at the end. Line 2680 initializes a bunch of variables so we can begin to process a field within the picture. (PICTURE.BUF is 256 bytes long. If you want a good project, figure out how to avoid using PICTURE.BUF. We could with more difficulty use the picture right where it is after AS.FRMEVL finishes.)

Lines 2700-2840 control the picture parsing. The basic idea is to scan through the picture executing command characters as we go, converting numbers to repeat counts, and printing literals. When a field descriptor is encountered, it is built up in WBUF to form a template for the conversion. If any of the characters of the descriptor were preceded by a repeat count, those characters will be reduplicated the specified number of times in the WBUF template. After the template is complete, an expression will be evaluated from the PRINT list, and converted into character form. Then those characters will be merged into the template, and the result printed. I got ahead of myself a little, but I wanted to give the overall view first.

PRUS.NEXT calls LOOKUP to process each character of the picture. Lookup searches the table shown in lines 3620-4010. Each entry in the table is three bytes long: the first byte is the character to be matched, and the next two are the address of a subroutine for processing that character. Actually this address is one less than the subroutine address, because it will be pushed onto the stack and branched to with an RTS instruction (see lines 3160-3190 and 3260). The order of the entries in the table is also somewhat significant. There are three groups of entries: the first group includes characters which may be part of a numberic field descriptor; the second, characters for string field descriptors; and the third, command characters. The labels L.EITHER and L.BOTH mark the edges of these three groups.

If LOOKUP matches a character, it checks to see if the character is in the third group (line 2980). If so, we know any field descriptor which may have been building is ended, so lines 3000-3010 clear the FLD.FLAG. If not, lines 3030-3070 start a new field unless we were already in one.

Lines 3080-3140 check if we have finished a field descriptor. We may have, if the matched character was a command character or a field-descriptor character of the opposite type field. So, if the mathced character was a numeric-field character, we call PRT.STR.IF.NEEDED; if it was a string-field character, we call PRT.NUM.IF.NEEDED; and if a command character, we call both of the IF.NEEDED's. The IF.NEEDED routines check if we were building up the corresponding field descriptor. If so, we

need to get a value from the PRINT list and print it now, before continuing to process the latest picture character.

Next, LOOKUP branches to the processor for the particular character matched. It sets up the repeat count, if any has been accumulated, in the Y-register. If no repeat count has been accumulated, y is set to 1. The routines are all in lines 4020-5250.

If LOOKUP does not find the picture character in the table, it may be a digit of a repeat count. If so, lines 3280-3450 multiply the existing repeat count by ten and add in the new digit. No check for overflow is done here, so if you write a repeat count of more than 255, it will be taken modulo 256. If you want to check for overflow, insert the check after line 3330:

```
CMP #25
BCS RP.OVERFLOW
```

and put a line after line 3610:

```
RP.OVERFLOW JMP AS.OVRFLW
```

If the character is not even a digit, it is good for nothing but separating field descriptors. Lines 3470-3480 call the two IF.NEEDED routines, in case a field descriptor preceded the non-matching character, and then fall into PRUS.CLEAR to get ready for the next picture character.

If the picture character is Z, #, or * the code at lines 4070-4240 goes to work. There are three different entry points here. A "Z" enters at IP.ZERO, where the A-register is cleared and a $2C opcode is used to skip over the following two bytes of code. You may recall that $2C is the opcode for BIT with a two-byte address. The 6502 acts like the "LDA #' '" is an address for the BIT instruction, and in effect that "hops over" line 4110. (This is a common coding trick in the 6502 world, and is safe except when the second of the two skipped bytes is in the range from $C0 through $C7. In that range you run the risk of flipping some soft switches in the I/O space.)

Lines 4070-4140 store zero, blank, or asterisk in FILL.CHAR and in the template being created in WBUF. These positions in the template will later be replaced with the actual digits of the converted number, unless they precede the most significant digit. The "w" and "d" parameters are also incremented as appropriate, so that we can later call FOUT to create the initial image of the converted number. Lines 4220-4230 loop on the repeat count, storing multiple copies of the fill character if you used a repeat count. We also set the FOUND.NUM flag non-zero, so that the PRT.NUM.IF.NEEDED subroutine will realize the need to print.

The RTS on the end of all the IP... processors takes control back to the middle of PRUS.NEXT, because they are actually just extensions to LOOKUP.

Lines 4290-4310 handle both the + and - picture characters. The character is stored in the template, and also in SIGN.CHAR1 as a flag. We need later to know whether any + or - appeared in the template at all, so the flag will be useful then.

If a decimal point appears in the picture, we store it in the template and also note the fact by setting DECFLG non-zero. A comma is merely stored in the template. See lines 4340-4440 for these two.

Lines 4450-4560 build templates for string field descriptors. The characters A, C, and R and just counted, while saving the lates one in FOUND.CHAR. When the PRT.STR.IF.NEEDED subroutine is called later, all we will need to know is which mode to use (A, C, or R) and how wide the field is.

Lines 4570-4760 print literal strings from the picture. The only tricky part of this is the handling of the closing apostrophe. A single apostrophe signals the end of the literal string, while two apostrophe's in a row mean an apostrophe should be printed within the literal.

Slash or "X" in a picture are handled by lines 4770-4880. Note the use again of the $2C to skip over two bytes of code.

Lines 4900-4960 handle the HTAB command. This is the bare minimum handling, and I can suggest some enhancements you might like to add here. You might want to check and be sure the value is between 1 and 40, giving an error message if out of range. You might want to adapt it to work with your particular printer and 80-column card combinations. Or 132-column Ultra-Term. It's up to you.

Lines 4970-5040 process the VTAB command, and here I do check for a valid line number. Of course, if you have an Ultra-Term set up for more than 24 lines you would want to change the limit in line 5000.

Lines 5050-5180 handle the screen clearing commands. A single ">" character calls MON.CLREOL to clear from the cursor to the end of the current line. If the following character in the picture is also a ">", MON.CLREOS is called instead.

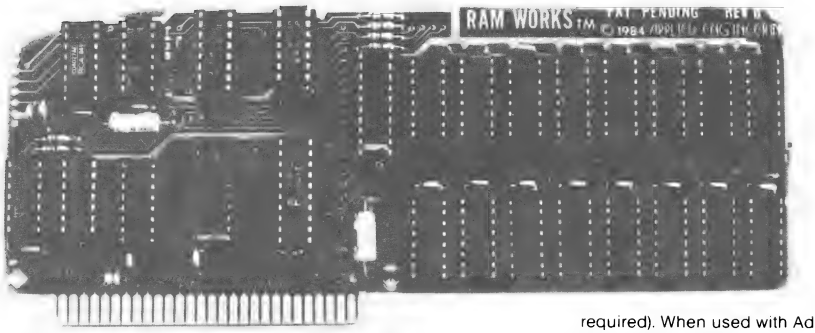
PRT.NUM.IF.NEEDED (lines 5190-5330)is one of the two IF.NEEDED twins. If FOUND.NUM is non-zero, indicating that we have been building a numeric field template, then now is the time to print a number. Unless, of course, we are doing INPUT$ rather than PRINT$. More on that subject next month.
PRT.STR.IF.NEEDED (lines 6320-6460) does the same for strings.

When a number needs to be printed, lines 5340-5420 get it ready for conversion. Line 5390 evaluates the next expression from the PRINT list, and it all falls into PRT.NUM.1 at line 5440. INPUT$ has an entry at this same point.

Lines 5450-5490 make room for the sign character if the expression value is negative and a sign reservation character was used in the template. Then W and D are correct for calling

RAMWORKS™

800K Apple Works Desk Top, 450K Visicalc, 1 Meg Solid State Disk
Double Hi-Res, RGB, Totally Apple Compatible!



RAMWORKS™—A card that plugs into the Apple IIe auxiliary slot and functions EXACTLY like Apple's extended 80 column card (in fact, a 128K RAMWORKS™ actually costs less than Apple's 64K extended card) but with RAMWORKS™ you get more memory, 80 column text, a 3-year warranty and most importantly, room to grow without using more slots. A design so advanced there's a patent pending on it. If you have a IIc or an IBM, we suggest you do what everybody does, trade it in on a IIe.

RAMWORKS™ can be purchased in a wide range of sizes and is user upgradeable using either 64K RAMS or the new 256K RAMS. In fact, RAMWORKS™ is the only auxiliary slot card on the market that will allow the new 256K RAMS to be used. If you already have an extended 80 column card, no problem. Just unplug the 64K RAMS and plug them into the RAMWORKS™ for an additional 64K. A RGB option is also available, you can order it with your RAMWORKS™ card or add it on at a later date.

RAMWORKS™ saves you time, money, slots and hassle. You'll have additional memory NOW and in the future.

Ramworks™

64K Installed	$ 179
128K Installed	$ 249
256K Installed	$ 399
512K Installed	$ 649
1 MEG Installed	$1199
RGB Option	$ 129

(May be added later)

LOW COST SOFTWARE OPTIONS

Ram Drive IIe

Ram Drive IIe will give you a high speed solid state disk drive. The Ram Drive IIe software features audio-visual access indicators, easy setup for turnkey operation, and easy menu driven documentation. The program can be modified and is copyable. If you have a 64K RAMWORKS™, Ram Drive IIe will act as half a disk drive. If you have a 128K or larger RAMWORKS™, Ram Drive IIe will act as a full disk drive. Either way, your programs will load and save over 20 times faster. Ram Drive IIe is compatible with APPLESOFT, PRO DOS, DOS 3.3, and PASCAL. The disk also includes a high speed RAMdisk copying program. Ram Drive is another disk drive only 20 times faster. And no whirring, clicking and waiting! **PRICE $29**

CP/M Ram Drive IIe

CP/M Ram Drive IIe is just like the Ram Drive IIe above, only for CP/M.

CP/M Ram Drive IIe runs on any Z-80 card that runs standard CP/M i.e. Applied Engineering Z-80 Plus or Microsoft Soft Card. CP/M Ram Drive will dramatically speed up the operation of most CP/M software because CP/M normally goes to disk fairly often. Fast acting software like dBase II, Wordstar and Turbo Pascal becomes virtually instantaneous when used with CP/M Ram Drive. **PRICE $29**

VC IIe Expander

VC IIe expander gives owners of Visicalc IIe and Advanced Visicalc IIe increased storage. When used with VC IIe you'll get 141K workspace (128K RAMWORKS™ or larger required). When used with Advanced VC IIe you'll get 131K with a 128K RAMWORKS™, 250K with a 256K RAMWORKS™ and 450K with a 512K RAMWORKS™. **PRICE $29**

Apple Works Expand

Although Apple Works is compatible with all sizes of RAMWORKS™, Apple Works only "sees" its first 64K bank giving you a 55K desktop. Our Apple Works expand program will make a modification to Apple Works that simply lets it know you've got more memory, giving you 101K work space. **PRICE $29**

Super Apple Works Expand

This souped-up version of Apple Works expand doesn't stop at a 101K desk top, in fact Super Apple Works Expand figures out how much memory your RAMWORKS™ has to give Apple Works the following desktop sizes:

RAMWORKS™	APPLEWORKS DESKTOP
128K	101K
256K	200K
512K	400K
1 MEG	800K

PRICE $39

AppleWorks can also be put in the RAMWORKS™ card to eliminate disk access, thereby dramatically speeding up the program.

APPLIED ENGINEERING

Send Check or Money Order to
Applied Engineering
P.O. Box 798, Carrollton, TX 75006
Call (214) 492-2027
8 a.m. to 11 p.m. 7 days a week. MasterCard, Visa & C.O.D. Welcome. No extra charge for credit cards. Texas residents add 5%-% sales tax. Add $10.00 if outside U.S.A.

FOUT in lines 5500-5530. The remainder of the PRINT.NUM subroutine copies characters from the FOUT.BUF string into the template, and then prints the fleshed-out template. Sounds easier than it really is....

Lines 5540-5690 control the scan through the template in WBUF. Commas in the template are handled right there: if any previous digits have been displayed, or if the fill character is "0" or "*", the comma is left in the template. If no digits have been stored yet and the fill character is blank, the comma is blanked out. It would look kind of silly hanging out in front of a number.

Lines 5700-5720 process a + or - character from the template. The actual code for PRUS.SGN at lines 6110-6310 does the work. If the template character is "+", it gets changed to "-" if the sign of the numeric value is negative. If the template character is "-", it gets changed to blank if the numeric value is positive.

If the template character is a digit place-holder, the next character from FOUT.BUF is examined. If the FOUT.BUF character is a digit, it is stored into the template. If not a digit, it might be a decimal point, a minus sign, or a leading blank. A leading blank gets changed to whatever the fill character is for the current template and stored in the template. A minus sign will be stored if there was no sign-position character in the template. A decimal point will be in the same position in both template and FOUT.BUF, so nothing needs to be done with it.

Since a sign-position character could come at the end of the template, lines 6000-6020 check for that condition.

Finally, lines 6030-6100 print out the composite string from WBUF.

String fields are printed by PRINT.STR, starting at line 6470. Lines 6470-6550 evaluate a string expression from the PRINT list, and set up a pointer to the resulting string value. The entry PRINT.STR.1 is shared with INPUT$. Lines 6570-6620 determine how much longer the field is than the string value. If it is too short, lines 6630-6700 fill the field with stars for an overflow indication.

If the string will fit, lines 6710-6750 store the number of left-over spaces in the field. If we are left-justifying, these will all come at the end; if right-justifying, at the beginning; if centering, half on each end. Lines 6760-6800 branch according to which type of string field we have (A, C, or R). Lines 6810-6840 print leading spaces for type-R fields.

Lines 6850-6910 divide the number of extra spaces in half, so half can be printed before the string and half after. If there were an odd number of extra spaces, the extra extra space will be printed after the string. For example, a four-character string in a nine-character field would be preceded by two blanks and followed by three.

That about winds up the discussion of the DP18 PRINT support. You can add or subtract features from this base, to create the exact configuration you need.

I should give credit to Bobby Deen for the original coding of the PRINT statement routines published this month, and the INPUT stuff next month. I revised them considerably since he wrote them two years ago, but you can still see his marks. Bobby is still pulling in a 4.0 average (highest possible) at Texas A & M, and programming for pay at the same time.

```
                1000 *SAVE S.DP18 PRINT
                1010 *--------------------------------
                1020 *    APPLESOFT SUBROUTINES
                1030 *--------------------------------
DAFB-           1040 AS.CROUT      .EQ $DAFB    PRINT CARRIAGE RETURN
DB5C-           1050 AS.COUT       .EQ $DB5C    PRINT A CHARACTER
DD7B-           1060 AS.FRMEVL     .EQ $DD7B    EVAL FP FORM. OR STRING
DEBE-           1070 AS.CHKCOM     .EQ $DEBE    CHECK FOR COMMA
DEC9-           1080 AS.SYNERR     .EQ $DEC9    SYNTAX ERROR
E199-           1090 AS.ILLERR     .EQ $E199    ILLEGAL QUANTITY ERROR
E5FD-           1100 AS.FRESTR     .EQ $E5FD   ERR IF NOT STRING,  FREE UP A TEMP
E6F5-           1110 AS.GTBYTC     .EQ $E6F5    CHRGET, THEN GETBYT        STRING
E6F8-           1120 AS.GETBYT     .EQ $E6F8    GET EXPR AS BYTE IN X
                1130 *--------------------------------
                1140 *   MONITOR SUBROUTINES
                1150 *--------------------------------
FC24-           1160 MON.VTABZ     .EQ $FC24
FC42-           1170 MON.CLREOS    .EQ $FC42
FC9C-           1180 MON.CLREOL    .EQ $FC9C
                1190 *--------------------------------
                1200 *      DP SUBROUTINES PRINTED ELSEWHERE
                1210 *--------------------------------
FFFF-           1220 DP.NEXT.CMD          .EQ $FFFF
FFFF-           1230 DP.EVALUATE          .EQ $FFFF
FFFF-           1240 FOUT                 .EQ $FFFF
FFFF-           1250 QUICK.PRINT          .EQ $FFFF
FFFF-           1260 FORMAT.PRINT         .EQ $FFFF
FFFF-           1270 INPUT.NUM            .EQ $FFFF
FFFF-           1280 INPUT.STR            .EQ $FFFF
                1290 *--------------------------------
                1300 *      PAGE ZERO USAGE
                1310 *--------------------------------
24-             1320 MON.CH        .EQ $24
25-             1330 MON.CV        .EQ $25
B1-             1340 AS.CHRGET     .EQ $B1
B7-             1350 AS.CHRGOT     .EQ $B7
F9-             1360 P2            .EQ $F9
FD-             1370 P1            .EQ $FD       GP POINTER
FB-             1380 TEMP2         .EQ $FB
                1390 *--------------------------------
0200-           1400 WBUF          .EQ $0200
                1410 *--------------------------------
                1420 *      WORK AREAS FOR DPFP
                1430 *--------------------------------
0800-           1440 DECFLG                .BS 1
0801-           1450 DAC.EXPONENT          .BS 1
0802-           1460 DAC.SIGN              .BS 1
0803-           1470 FOUT.BUF              .BS 41
082C-           1480 STACK.PNTR            .BS 1
082D-           1490 W                     .BS 1
082E-           1500 D                     .BS 1
082F-           1510 SIGN.CHAR1            .BS 1
0830-           1520 INPUT.TYPE            .BS 1
0831-           1530 FOUND.NUM             .BS 1
0832-           1540 FOUND.STR             .BS 1
0833-           1550 STR.LEN               .BS 1
0834-           1560 REPEAT.CNT            .BS 1
0835-           1570 FOUND.LEN             .BS 1
0836-           1580 FOUND.CHAR            .BS 1
0837-           1590 FILL.CHAR             .BS 1
0838-           1600 CHAR                  .BS 1
0839-           1610 INPUT.FLAG            .BS 1
083A-           1620 ZERO.CHAR             .BS 1
```

```
083B-                1630 FLD.FLAG            .BS 1
083C-                1640 FLD.START           .BS 1
083D-                1650 TEMP3               .BS 2
083F-                1660 INDEX               .BS 1
0840-                1670 PICTURE.BUF         .BS 256
                     1680 *-------------------------------
                     1690 DP.PRINT
0940- 20 B1 00       1700         JSR AS.CHRGET
0943- 20 8E 09       1710         JSR PRINT.END
0946- 4C FF FF       1720         JMP DP.NEXT.CMD
                     1730 *-------------------------------
                     1740 DP.UNFORMAT
0949- 20 FF FF       1750         JSR DP.EVALUATE    GET EXPRESSION
094C- AD 01 08       1760         LDA DAC.EXPONENT   GET EXPONENT
094F- C9 53          1770         CMP #$40+19        MORE THAN 18 DIGITS BEFORE DECPT?
0951- B0 38          1780         BCS .5             YES, USE SCIENTIFIC
0953- C9 3F          1790         CMP #$40-1         LESS THAN .01?
0955- 90 34          1800         BCC .5             YES, USE SCIENTIFIC
0957- A9 30          1810         LDA #'0
0959- 8D 3A 08       1820         STA ZERO.CHAR
095C- A9 28          1830         LDA #40            ALLOW PLENTY OF WIDTH
095E- A0 13          1840         LDY #19            AND DECIMAL PLACES
0960- 20 FF FF       1850         JSR FOUT
                     1860 *---TRIM TRAILING ZEROES---------
0963- AC 3F 08       1870         LDY INDEX          FIND END OF BUFFER
0966- 88             1880 .1      DEY
0967- B9 02 08       1890         LDA FOUT.BUF-1,Y   TRUNCATE TRAILING ZEROES
096A- C9 30          1900         CMP #'0            IS THIS ONE ZERO?
096C- F0 F8          1910         BEQ .1             ...YES, KEEP TRIMMING
096E- C9 2E          1920         CMP #'.            OMIT DECIMAL POINT ON INTEGERS
0970- F0 01          1930         BEQ .2             ...GOT A DECPT
0972- C8             1940         INY                TRIM NO MORE...
0973- A9 00          1950 .2      LDA #0             MARK END OF MEANINGFUL CHARS
0975- 99 02 08       1960         STA FOUT.BUF-1,Y
0978- 8C 3F 08       1970         STY INDEX
                     1980 *---PRINT WITHOUT LEADING BLANKS-
097B- A8             1990         TAY                Y=0
097C- B9 03 08       2000 .3      LDA FOUT.BUF,Y
097F- F0 0D          2010         BEQ PRINT.END
0981- C9 20          2020         CMP #$20           BLANK?
0983- F0 03          2030         BEQ .4             ...YES, DON'T PRINT
0985- 20 5C DB       2040         JSR AS.COUT        ...NO, PRINT IT
0988- C8             2050 .4      INY
0989- D0 F1          2060         BNE .3             ...ALWAYS
                     2070 *---PRINT WITH EXPONENT----------
098B- 20 FF FF       2080 .5      JSR QUICK.PRINT
                     2090 *-------------------------------
                     2100 PRINT.END
098E- 20 B7 00       2110         JSR AS.CHRGOT
0991- D0 03          2120         BNE .1             NOT ":" OR EOL
0993- 4C FB DA       2130         JMP AS.CROUT
0996- C9 3B          2140 .1      CMP #';'
0998- F0 2B          2150         BEQ .3
099A- C9 2C          2160         CMP #','
099C- F0 24          2170         BEQ .2
099E- C9 24          2180         CMP #'$       PRINT USING?
09A0- F0 29          2190         BEQ DP.PRINT.USING
09A2- C9 23          2200         CMP #'#       PRINT W,D?
09A4- D0 A3          2210         BNE DP.UNFORMAT NO,UNFORMATTED PRINT
                     2220 *---PRINT #W,D,VALUE-------------
09A6- 20 F5 E6       2230         JSR AS.GTBYTC      GET W IN X-REG
09A9- 8A             2240         TXA
09AA- 48             2250         PHA
09AB- 20 BE DE       2260         JSR AS.CHKCOM      MUST HAVE COMMA
09AE- 20 F8 E6       2270         JSR AS.GETBYT      GET D IN X-REG
09B1- 8A             2280         TXA
09B2- 48             2290         PHA
09B3- 20 BE DE       2300         JSR AS.CHKCOM      ANOTHER COMMA
09B6- 20 FF FF       2310         JSR DP.EVALUATE    GET EXPR
09B9- 68             2320         PLA                GET D
09BA- A8             2330         TAY
09BB- 68             2340         PLA                GET W
09BC- 20 FF FF       2350         JSR FORMAT.PRINT
09BF- 4C 8E 09       2360         JMP PRINT.END
                     2370 *---COMMA AFTER ITEM-------------
09C2- 20 FB DA       2380 .2      JSR AS.CROUT       DP18'S KIND OF TABBING
```

```
                        2390 *---",* OR ":* AFTER ITEM--------
09C5- 20 B1 00 2400 .3      JSR AS.CHRGET       NEXT CHAR
09C8- D0 CC    2410         BNE .1              NEXT PRINT ITEM
09CA- 60       2420         RTS
               2430 *--------------------------------
               2440 DP.PRINT.USING
09CB- A9 01    2450         LDA #1         PRINT,NOT INPUT
               2460 *--------------------------------
               2470 PRINT.INPUT
09CD- 8D 39 08 2480         STA INPUT.FLAG      0=INPUT, 1=PRINT
09D0- 20 B1 00 2490         JSR AS.CHRGET       EAT THE $
09D3- 20 7B [illegible] 2500         JSR AS.FRMEVL       GET PICTURE
09D6- 20 FD [illegible] 2510         JSR AS.FRESTR       ERR IF NOT STRING, FREE TEMP
09D9- 86 FD    2520         STX P1                ADDR IN Y,X, LEN IN A
09DB- 84 FE    2530         STY P1+1
09DD- 8D 33 08 2540         STA STR.LEN
09E0- EE 33 08 2550         INC STR.LEN         WE'RE GOING TO ADD ONE
09E3- A8       2560         TAY                 LENGTH TO Y
09E4- A9 00    2570         LDA #0              PUT 0 AT END OF PICTURE
09E6- 99 40 08 2580         STA PICTURE.BUF,Y
09E9- 8D 2C 08 2590         STA STACK.PNTR
09EC- 8D 3B 08 2600         STA FLD.FLAG
09EF- 88       2610 .1      DEY
09F0- B1 FD    2620         LDA (P1),Y          MOVE PICTURE TO BUFFER
09F2- 99 40 08 2630         STA PICTURE.BUF,Y
09F5- 98       2640         TYA                 TEST FOR END
09F6- D0 F7    2650         BNE .1              ...MORE
09F8- 8C 34 08 2660         STY REPEAT.CNT      Y IS 0
09FB- 88       2670         DEY                 Y = $FF
09FC- 20 9B 0A 2680         JSR PRUS.CLEAR      CLEAR VARIABLES
               2690 *--------------------------------
               2700 *   PARSE THE PICTURE
               2710 *--------------------------------
               2720 PRUS.NEXT
09FF- C8       2730         INY                 NEXT CHAR
0A00- CC 33 08 2740         CPY STR.LEN      DONE?
0A03- F0 0D    2750         BEQ .1              ...YES
0A05- B9 40 08 2760         LDA PICTURE.BUF,Y GET A CHAR
0A08- 84 FB    2770         STY TEMP2           SAVE PICTURE PNTR
0A0A- 20 1D 0A 2780         JSR LOOKUP
0A0D- A4 FB    2790         LDY TEMP2           RESTORE PICTURE PNTR
0A0F- 4C FF 09 2800         JMP PRUS.NEXT
0A12- AD 39 08 2810 .1      LDA INPUT.FLAG
0A15- D0 03    2820         BNE .2
0A17- 4C FB DA 2830         JMP AS.CROUT
0A1A- 4C 8E 09 2840 .2      JMP PRINT.END       HANDLE ; AT END OF STATEMENT
               2850 *--------------------------------
               2860 * LOOKUP LOOKS UP THE ENTRY CORRESPONDING TO (A)
               2870 *--------------------------------
0A1D- 8D 38 08 2880 LOOKUP  STA CHAR        SAVE KEY
0A20- A0 FD    2890         LDY #-3
0A22- C8       2900 .1      INY
0A23- C8       2910         INY
0A24- C8       2920         INY                 NEXT ENTRY
0A25- B9 B7 0A 2930         LDA TBL.BASE,Y
0A28- F0 42    2940         BEQ .7              END OF TABLE
0A2A- CD 38 08 2950         CMP CHAR            ONE WE WANT?
0A2D- D0 F3    2960         BNE .1              NO,NEXT ENTRY
               2970 *---FOUND CHAR IN TABLE----------
0A2F- C0 1E    2980         CPY #L.BOTH         NEW FIELD?
0A31- 90 07    2990         BCC .2              ...MAYBE NOT
0A33- A9 00    3000         LDA #0              START A NEW FIELD
0A35- 8D 3B 08 3010         STA FLD.FLAG
0A38- F0 0D    3020         BEQ .3              ...ALWAYS
0A3A- AD 3B 08 3030 .2      LDA FLD.FLAG BEGINNING OF FIELD?
0A3D- D0 08    3040         BNE .3              NO
0A3F- A5 FB    3050         LDA TEMP2
0A41- 8D 3C 08 3060         STA FLD.START
0A44- EE 3B 08 3070         INC FLD.FLAG
               3080 *---PRINT WHATEVER'S NEEDED------
0A47- C0 15    3090 .3      CPY #L.EITHER
0A49- 90 07    3100         BCC .4              ...ONLY TRY PRT.STR.IF.NEEDED
0A4B- 20 72 0B 3110         JSR PRT.NUM.IF.NEEDED
0A4E- C0 1E    3120         CPY #L.BOTH
0A50- 90 03    3130         BCC .5              ...ONLY TRY PRT.NUM.IF.NEEDED
0A52- 20 4F 0C 3140 .4      JSR PRT.STR.IF.NEEDED
               3150 *---GET ROUTINE ADDRESS----------
0A55- B9 B9 0A 3160 .5      LDA TBL.BASE+2,Y
0A58- 48       3170         PHA                 PUT ADDRESS ON STACK
0A59- B9 B8 0A 3180         LDA TBL.BASE+1,Y
```

```
0A5C- 48          3190         PHA
0A5D- AC 34 08    3200         LDY REPEAT.CNT     GET THE COUNT
0A60- D0 01       3210         BNE .6             COUNT IS NON-0
0A62- C8          3220         INY                COUNT IS 0, SO MAKE IT 1
0A63- A9 00       3230 .6      LDA #0             CLEAR REPEAT.CNT
0A65- 8D 34 08    3240         STA REPEAT.CNT
0A68- AD 38 08    3250         LDA CHAR           GET THE ORIGINAL CHARACTER
0A6B- 60          3260         RTS                JUMP TO ROUTINE
                  3270 *---CHAR NOT IN TABLE-----------
0A6C- AD 38 08    3280 .7      LDA CHAR           GET CHAR AGAIN
0A6F- 49 30       3290         EOR #'0            CHECK FOR DIGIT 0-9
0A71- C9 0A       3300         CMP #10
0A73- B0 20       3310         BCS .9             ...NOT A NUMBER
0A75- 8D 3D 08    3320         STA TEMP3
0A78- AD 34 08    3330         LDA REPEAT.CNT     PREVIOUS * 10
0A7B- 0A          3340         ASL                *2
0A7C- 0A          3350         ASL                *4
0A7D- 6D 34 08    3360         ADC REPEAT.CNT     *5
0A80- 0A          3370         ASL                *10
0A81- 6D 3D 08    3380         ADC TEMP3          + DIGIT
0A84- 8D 34 08    3390         STA REPEAT.CNT
0A87- AD 3B 08    3400         LDA FLD.FLAG       BEGINNING OF FIELD?
0A8A- D0 08       3410         BNE .8             ...NO
0A8C- A5 FB       3420         LDA TEMP2          YES, SAVE STARTING POSN
0A8E- 8D 3C 08    3430         STA FLD.START
0A91- EE 3B 08    3440         INC FLD.FLAG
0A94- 60          3450 .8      RTS
                  3460 *---NOT IN TABLE, NOT A DIGIT----
0A95- 20 4F 0C    3470 .9      JSR PRT.STR.IF.NEEDED
0A98- 20 72 0B    3480         JSR PRT.NUM.IF.NEEDED
                  3490 *-------------------------------
                  3500 PRUS.CLEAR
0A9B- A2 01       3510         LDX #1
0A9D- 8E 2D 08    3520         STX W              W = 1
0AA0- CA          3530         DEX                REST = 0
0AA1- 8E 2E 08    3540         STX D
0AA4- 8E 00 08    3550         STX DECFLG         NO DECIMAL
0AA7- 8E 2F 08    3560         STX SIGN.CHAR1
0AAA- 8E 31 08    3570         STX FOUND.NUM      FLAG IF # HAS BEEN FOUND
0AAD- 8E 32 08    3580         STX FOUND.STR
0AB0- 8E 35 08    3590         STX FOUND.LEN
0AB3- 8E 36 08    3600         STX FOUND.CHAR
0AB6- 60          3610         RTS
                  3620 *-------------------------------
                  3630 *    TABLE IS IN THREE SECTIONS:
                  3640 *       1ST SECTION (BEFORE L.EITHER) ARE FOR
                  3650 *       FOR DESCRIBING NUMERIC FIELDS, AND CAN
                  3660 *       TERMINATE A STRING FIELD.
                  3670 *
                  3680 *       2ND SECTION (BTWN L.EITHER & L.BOTH) IS
                  3690 *       FOR DESCRIBING STRING FIELDS, AND CAN
                  3700 *       TERMINATE A NUMERIC FIELD
                  3710 *
                  3720 *       3RD SECTION (AFTER L.BOTH) CAN TERMINATE
                  3730 *        BOTH KINDS OF FIELDS.
                  3740 *
                  3750 *       TABLE FORMAT = #CHAR,ADDRESS-1
                  3760 *       END OF TABLE MARKED WITH $00
                  3770 *-------------------------------
                  3780         .MA TBL
                  3790         .DA #']1',]2-1
                  3800         .EM
                  3810 *-------------------------------
                  3820 TBL.BASE
0AB7-             3830         >TBL "+",IP.PLUS.MINUS   -#-
0AB7- 2B 06 0B    0000>         .DA #'+',IP.PLUS.MINUS-1
0ABA-             3840         >TBL "-",IP.PLUS.MINUS   -#-
0ABA- 2D 06 0B    0000>         .DA #'-',IP.PLUS.MINUS-1
0ABD-             3850         >TBL "#",IP.NUMBER       -#-
0ABD- 23 EA 0A    0000>         .DA #'#',IP.NUMBER-1
0AC0-             3860         >TBL "*",IP.ASTERISK     -#-
0AC0- 2A EC 0A    0000>         .DA #'*',IP.ASTERISK-1
0AC3-             3870         >TBL "Z",IP.ZERO         -#-
0AC3- 5A E7 0A    0000>         .DA #'Z',IP.ZERO-1
0AC6-             3880         >TBL ".",IP.POINT        -#-
0AC6- 2E 0C 0B    0000>         .DA #'.',IP.POINT-1
0AC9-             3890         >TBL ",",IP.COMMA        -#-
0AC9- 2C 0F 0B    0000>         .DA #',',IP.COMMA-1
```

```
15-             3900 L.EITHER .EQ *-TBL.BASE
0ACC-           3910          >TBL "A",IP.ACR          -$-
0ACC- 41 14 0B  0000>          .DA #'A'-1,IP.ACR-1
0ACF-           3920          >TBL "C",IP.ACR          -$-
0ACF- 43 14 0B  0000>          .DA #'C'-1,IP.ACR-1
0AD2-           3930          >TBL "R",IP.ACR          -$-
0AD2- 52 14 0B  0000>          .DA #'R'-1,IP.ACR-1
1E-             3940 L.BOTH    .EQ *-TBL.BASE
0AD5-           3950          >TBL "'",IP.QT           -#$-
0AD5- 27 23 0B  0000>          .DA #''-1,IP.QT-1
0AD8-           3960          >TBL "/",IP.SLASH        -#$-
0AD8- 2F 44 0B  0000>          .DA #'/'-1,IP.SLASH-1
0ADB-           3970          >TBL "X",IP.X            -#$-
0ADB- 58 47 0B  0000>          .DA #'X'-1,IP.X-1
0ADE-           3980          >TBL "H",IP.HTAB         -#$-
0ADE- 48 50 0B  0000>          .DA #'H'-1,IP.HTAB-1
0AE1-           3990          >TBL "V",IP.VTAB         -#$-
0AE1- 56 54 0B  0000>          .DA #'V'-1,IP.VTAB-1
0AE4-           4000          >TBL ">",IP.GREATER      -#$-
0AE4- 3E 60 0B  0000>          .DA #'>'-1,IP.GREATER-1
0AE7- 00        4010          .HS 00        END OF TABLE
                4020 *--------------------------------
                4030 *   Z -- Digit position marker, zero fill
                4040 *   # -- Digit position marker, blank fill
                4050 *   * -- Digit position marker, star fill
                4060 *--------------------------------
                4070 IP.ZERO
0AE8- A9 30     4080          LDA #'0       USE 0 FOR FILL CHAR
0AEA- 2C        4090          .HS 2C
                4100 IP.NUMBER
0AEB- A9 20     4110          LDA #' '      USE BLANK FOR FILL CHAR
                4120 IP.ASTERISK
0AED- 8D 37 08  4130          STA FILL.CHAR  SAVE AS FILL CHAR
0AF0- 20 10 0B  4140 .1       JSR STA.WBUFX.INX
0AF3- EE 31 08  4150          INC FOUND.NUM  FOUND A DIGIT
0AF6- EE 2D 08  4160          INC W          LENGTH
0AF9- 48        4170          PHA
0AFA- AD 00 08  4180          LDA DECFLG     HAD DECIMAL PT?
0AFD- F0 03     4190          BEQ .2         NO
0AFF- EE 2E 08  4200          INC D          YES
0B02- 68        4210 .2       PLA
0B03- 88        4220          DEY
0B04- D0 EA     4230          BNE .1         NEXT ONE
0B06- 60        4240          RTS
                4250 *--------------------------------
                4260 *   + -- Sign position marker (prints + or -)
                4270 *   - -- Sign position marker (prints apace or -)
                4280 *--------------------------------
                4290 IP.PLUS.MINUS
0B07- 8D 2F 08  4300          STA SIGN.CHAR1 SAVE SIGN CHAR
0B0A- 4C 10 0B  4310          JMP STA.WBUFX.INX
                4320 *--------------------------------
                4330 *   . -- Decimal position marker
                4340 *--------------------------------
                4350 IP.POINT
0B0D- EE 00 08  4360          INC DECFLG     FOUND A DECIMAL POINT
                4370 *--------------------------------
                4380 *   , -- Puts a comma in a number
                4390 *--------------------------------
                4400 IP.COMMA
                4410 STA.WBUFX.INX
0B10- 9D 00 02  4420          STA WBUF,X     SAVE CHAR
0B13- E8        4430          INX
0B14- 60        4440          RTS
                4450 *--------------------------------
                4460 *   A -- String field, left justified
                4470 *   C -- String field, centered
                4480 *   R -- String field, right justified
                4490 *--------------------------------
0B15- EE 32 08  4500 IP.ACR INC FOUND.STR      FOUND A STRING
0B18- 8D 36 08  4510          STA FOUND.CHAR     SAVE THE CHAR
0B1B- 98        4520          TYA
0B1C- 18        4530          CLC
0B1D- 6D 35 08  4540          ADC FOUND.LEN  ADD LENGTH TO REPEAT COUNT
0B20- 8D 35 08  4550          STA FOUND.LEN
0B23- 60        4560          RTS
```

```
                    4570 *--------------------------------
                    4580 *   ' -- Start of embedded string
                    4590 *--------------------------------
                    4600 IP.QT
0B24- A6 FB         4610 .1     LDX TEMP2           X = PICTURE PNTR
0B26- E8            4620 .2     INX
0B27- BD 40 08      4630        LDA PICTURE.BUF,X GET CHAR
0B2A- C9 27         4640        CMP #''           APOSTROPHE?
0B2C- D0 08         4650        BNE .3            ...NO, PRINT IT
0B2E- BD 41 08      4660        LDA PICTURE.BUF+1,X
0B31- C9 27         4670        CMP #''           TWO APOSTROPHE'S IN A ROW?
0B33- D0 07         4680        BNE .4            ...NO, MEANS END OF LITERAL
0B35- E8            4690        INX               ...YES, PRINT APOSTROPHE
0B36- 20 5C DB      4700 .3     JSR AS.COUT
0B39- 4C 26 0B      4710        JMP .2
0B3C- 88            4720 .4     DEY           REPEAT COUNT
0B3D- D0 E5         4730        BNE .1        ...REPEAT THE STRING
0B3F- 86 FB         4740        STX TEMP2     NEW PICTURE PNTR
0B41- 60            4750        RTS
0B42- 4C C9 DE      4760 .5     JMP AS.SYNERR
                    4770 *--------------------------------
                    4780 *   / -- Print n carriage returns
                    4790 *   X -- print n spaces
                    4800 *--------------------------------
                    4810 IP.SLASH
0B45- A9 0D         4820        LDA #$0D      CR'S
0B47- 2C            4830        .HS 2C        (SKIP NEXT 2 BYTES)
0B48- A9 20         4840 IP.X   LDA #$20      BLANKS'
0B4A- 20 5C DB      4850 .1     JSR AS.COUT   PRINT THE CHAR
0B4D- 88            4860        DEY
0B4E- D0 FA         4870        BNE .1
0B50- 60            4880        RTS
                    4890 *--------------------------------
                    4900 *   H -- HTAB to column n
                    4910 *   V -- VTAB to line n
                    4920 *--------------------------------
                    4930 IP.HTAB
0B51- 88            4940        DEY
0B52- 84 24         4950        STY MON.CH    HTAB
0B54- 60            4960        RTS
                    4970 *--------------------------------
                    4980 IP.VTAB
0B55- 88            4990        DEY
0B56- C0 18         5000        CPY #24
0B58- B0 04         5010        BCS .1        OUT OF RANGE
0B5A- 98            5020        TYA
0B5B- 4C E1 0C      5030        JMP DP.VTAB
0B5E- 4C 99 E1      5040 .1     JMP AS.ILLERR  ILLEGAL QUANTITY ERROR
                    5050 *--------------------------------
                    5060 *   > -- CLEAR TO END OF LINE
                    5070 *  >> -- CLEAR TO END OF SCREEN
                    5080 *--------------------------------
                    5090 IP.GREATER
0B61- A4 FB         5100        LDY TEMP2
0B63- B9 41 08      5110        LDA PICTURE.BUF+1,Y
0B66- C9 3E         5120        CMP #'>'
0B68- F0 03         5130        BEQ .1        ...CLEAR TO END OF SCREEN
                    5140 *---CLEAR TO END OF LINE---------
0B6A- 4C 9C FC      5150        JMP MON.CLREOL
                    5160 *---CLEAR TO END OF SCREEN-------
0B6D- E6 FB         5170 .1     INC TEMP2
0B6F- 4C 42 FC      5180        JMP MON.CLREOS
                    5190 *--------------------------------
                    5200 PRT.NUM.IF.NEEDED
0B72- AD 31 08      5210        LDA FOUND.NUM  HAS # BEEN FOUND?
0B75- F0 15         5220        BEQ .1        NO
0B77- 98            5230        TYA
0B78- 48            5240        PHA           SAVE Y
0B79- AD 39 08      5250        LDA INPUT.FLAG
0B7C- F0 06         5260        BEQ .2            INPUT
0B7E- 20 8D 0B      5270        JSR PRINT.NUM     PRINT
0B81- 4C 87 0B      5280        JMP .3
0B84- 20 FF FF      5290 .2     JSR INPUT.NUM
0B87- 68            5300 .3     PLA           RESTORE Y
0B88- A8            5310        TAY
0B89- 20 9B 0A      5320        JSR PRUS.CLEAR
0B8C- 60            5330 .1     RTS
```

```
                    5340 *--------------------------------
                    5350 PRINT.NUM
0B8D- A9 00         5360         LDA #0        PUT $00
0B8F- 9D 00 02      5370         STA WBUF,X    AT END OF STRING
0B92- 20 BE DE      5380         JSR AS.CHKCOM  MUST HAVE COMMA
0B95- 20 FF FF      5390         JSR DP.EVALUATE   GET EXPRESSION
0B98- A9 30         5400         LDA #'0
0B9A- 8D 3A 08      5410         STA ZERO.CHAR
                    5420 *
                    5430 *--------------------------------
                    5440 PRT.NUM.1
0B9D- AD 02 08      5450         LDA DAC.SIGN
0BA0- 10 08         5460         BPL .1
0BA2- AD 2F 08      5470         LDA SIGN.CHAR1  SIGN IS -
0BA5- F0 03         5480         BEQ .1        NO SIGN CHAR
0BA7- EE 2D 08      5490         INC W         RESERVE PLACE FOR SIGN
                    5500 *---CONVERT VALUE INTO FOUT.BUF--
0BAA- AD 2D 08      5510 .1      LDA W
0BAD- AC 2E 08      5520         LDY D
0BB0- 20 FF FF      5530         JSR FOUT
                    5540 *---FILL IN THE PICTURE----------
0BB3- A2 00         5550         LDX #0        INDEX INTO WBUF
0BB5- A0 00         5560         LDY #0        INDEX INTO FBUF
0BB7- 8C 00 08      5570         STY DECFLG    USE FOR DIGITS FLAG
0BBA- BD 00 02      5580 .2      LDA WBUF,X    GET CHAR FROM PICTURE
0BBD- F0 5D         5590         BEQ .10       END OF PICTURE
0BBF- C9 2C         5600         CMP #',       COMMA?
0BC1- D0 12         5610         BNE .3
0BC3- E8            5620         INX
0BC4- AD 00 08      5630         LDA DECFLG    ANY DIGITS BEFORE THIS?
0BC7- D0 F1         5640         BNE .2              ...YES, LEAVE COMMA
0BC9- AD 37 08      5650         LDA FILL.CHAR       ...NO, BUT LEAVE IF FILL
0BCC- C9 20         5660         CMP #' '                 IS NON-BLANK.
0BCE- D0 EA         5670         BNE .2         ...NOT BLANK, SO LEAVE IN THE COMMA
0BD0- 9D FF 01      5680         STA WBUF-1,X   ...COVER COMMA WITH BLANK
0BD3- D0 E5         5690         BNE .2         ...ALWAYS
                    5700 *---CHECK FOR PICTURE SIGN-------
0BD5- 20 2A 0C      5710 .3      JSR PRUS.SGN       IF + OR -, PROCESS
0BD8- 90 E0         5720         BCC .2             ...WAS + OR -
                    5730 *---PICTURE IS DIGIT OR DECPT----
0BDA- B9 03 08      5740         LDA FOUT.BUF,Y     GET CHAR FROM VALUE STRING
0BDD- C9 20         5750         CMP #$20           SPACE?
0BDF- D0 03         5760         BNE .5             ...NO
0BE1- AD 37 08      5770         LDA FILL.CHAR      ...YES, USE FILL CHAR
0BE4- 48            5780 .5      PHA                SAVE FOUT OR FILL CHAR
0BE5- C9 2D         5790         CMP #'-            IS IT A SIGN CHAR?
0BE7- D0 15         5800         BNE .7             ...NO
0BE9- AD 2F 08      5810         LDA SIGN.CHAR1     IS THERE A SIGN IN FORMAT?
0BEC- D0 13         5820         BNE .8             ...YES, SKIP THE SIGN
0BEE- BD 01 02      5830         LDA WBUF+1,X       ...NO, INSTALL SIGN HERE
0BF1- C9 2C         5840         CMP #',            (UNLESS NEXT PIC.CHAR IS COMMA)
0BF3- D0 06         5850         BNE .6             ...NOT COMMA
0BF5- AD 37 08      5860         LDA FILL.CHAR      ...COMMA, SO COVER WITH FILLER
0BF8- 20 10 0B      5870         JSR STA.WBUFX.INX
0BFB- B9 03 08      5880 .6      LDA FOUT.BUF,Y     GET SIGN CHAR AGAIN
0BFE- 20 10 0B      5890 .7      JSR STA.WBUFX.INX
0C01- 68            5900 .8      PLA                GET FOUT OR FILL CHAR BACK
0C02- C8            5910         INY                ADVANCE FOUT PNTR
0C03- CC 3F 08      5920         CPY INDEX          END OF FOUTBUF?
0C06- B0 0E         5930         BCS .9             ...YES
0C08- CD 37 08      5940         CMP FILL.CHAR      IF WE INSTALLED A DIGIT
0C0B- F0 AD         5950         BEQ .2             WE MUST SET THE DIGITS FLAG
0C0D- C9 2D         5960         CMP #'-            SIGN CHAR?
0C0F- F0 A9         5970         BEQ .2             ...YES
0C11- EE 00 08      5980         INC DECFLG         FOUND A DIGIT
0C14- D0 A4         5990         BNE .2             ...ALWAYS
                    6000 *---END OF FOUT.BUF--------------
0C16- BD 00 02      6010 .9      LDA WBUF,X
0C19- 20 2A 0C      6020         JSR PRUS.SGN
                    6030 *---END OF FOUT OR PICTURE-------
0C1C- A0 00         6040 .10     LDY #0
0C1E- B9 00 02      6050 .11     LDA WBUF,Y
0C21- F0 06         6060         BEQ .12
0C23- 20 5C DB      6070         JSR AS.COUT   PRINT IT
0C26- C8            6080         INY
0C27- D0 F5         6090         BNE .11       ALWAYS
0C29- 60            6100 .12     RTS
                    6110 *--------------------------------
```

```
                      6120 PRUS.SGN
0C2A- C9 2B           6130         CMP #'+       SIGN?
0C2C- D0 0D           6140         BNE .1        NO
0C2E- E8              6150         INX
0C2F- AD 02 08        6160         LDA DAC.SIGN
0C32- 10 17           6170         BPL .2        SIGN ALREADY +
0C34- A9 2D           6180         LDA #'-
0C36- 9D FF 01        6190         STA WBUF-1,X
0C39- D0 10           6200         BNE .2        ALWAYS
0C3B- C9 2D           6210 .1      CMP #'-       -?
0C3D- D0 0E           6220         BNE .3        NO
0C3F- E8              6230         INX
0C40- AD 02 08        6240         LDA DAC.SIGN
0C43- 30 06           6250         BMI .2        SIGN ALREADY -
0C45- AD 37 08        6260         LDA FILL.CHAR
0C48- 9D FF 01        6270         STA WBUF-1,X  BLANK OUT SIGN
0C4B- 18              6280 .2      CLC
0C4C- 60              6290         RTS
0C4D- 38              6300 .3      SEC
0C4E- 60              6310         RTS
                      6320 *-------------------------------
                      6330 PRT.STR.IF.NEEDED
0C4F- AD 32 08        6340         LDA FOUND.STR HAS STRING BEEN FOUND?
0C52- F0 15           6350         BEQ .3        NO
0C54- 98              6360         TYA
0C55- 48              6370         PHA           SAVE Y
0C56- AD 39 08        6380         LDA INPUT.FLAG
0C59- F0 06           6390         BEQ .1
0C5B- 20 6A 0C        6400         JSR PRINT.STR
0C5E- 4C 64 0C        6410         JMP .2
0C61- 20 FF FF        6420 .1      JSR INPUT.STR
0C64- 68              6430 .2      PLA
0C65- A8              6440         TAY           RESTORE Y
0C66- 20 9B 0A        6450         JSR PRUS.CLEAR
0C69- 60              6460 .3      RTS
                      6470 *-------------------------------
                      6480 PRINT.STR
0C6A- A9 20           6490         LDA #$20
0C6C- 8D 37 08        6500         STA FILL.CHAR
0C6F- 20 BE DE        6510         JSR AS.CHKCOM  MUST HAVE COMMA
0C72- 20 7B DD        6520         JSR AS.FRMEVL     GET EXPRESSION
0C75- 20 FD E5        6530         JSR AS.FRESTR     GET ADR AND LEN
0C78- 86 F9           6540         STX P2
0C7A- 84 FA           6550         STY P2+1
                      6560 *-------------------------------
                      6570 PRINT.STR.1
0C7C- 48              6580         PHA           SAVE LENGTH
0C7D- 38              6590         SEC              LENGTH IS IN A
0C7E- ED 35 08        6600         SBC FOUND.LEN    SUBTRACT FIELD LENGTH
0C81- F0 0F           6610         BEQ .2           ...SAME, SO OKAY
0C83- 90 0D           6620         BCC .2           ...EXP IS SHORTER THAN FIELD
                      6630 *---FIELD OVERFLOW---------------
0C85- 68              6640         PLA              DISCARD LENGTH
0C86- AC 35 08        6650         LDY FOUND.LEN    GET FIELD LEN
0C89- A9 2A           6660         LDA #'*          OVERFLOW CHAR
0C8B- 20 5C DB        6670 .1      JSR AS.COUT
0C8E- 88              6680         DEY
0C8F- D0 FA           6690         BNE .1
0C91- 60              6700         RTS
                      6710 *---JUSTIFY IN FIELD-------------
0C92- 49 FF           6720 .2      EOR #$FF      GET POSITIVE #
0C94- A8              6730         TAY
0C95- C8              6740         INY
0C96- 8C 35 08        6750         STY FOUND.LEN
0C99- AD 36 08        6760         LDA FOUND.CHAR
0C9C- C9 41           6770         CMP #'A       LJ FIELD
0C9E- F0 16           6780         BEQ .5
0CA0- C9 43           6790         CMP #'C       CJ FIELD
0CA2- F0 07           6800         BEQ .4
                      6810 *---RIGHT JUSTIFY----------------
0CA4- 20 D4 0C        6820         JSR PRINT.Y.SPACES
0CA7- 68              6830         PLA           RESTORE STRING LEN
0CA8- 4C C0 0C        6840         JMP PRT.STR   PRINT STRING
                      6850 *---CENTER JUSTIFY---------------
0CAB- 98              6860 .4      TYA           # OF SPACES
0CAC- 4A              6870         LSR           DIVIDE BY 2
0CAD- A8              6880         TAY           # LEADING BLANKS
0CAE- 69 00           6890         ADC #0        +1 IF IT WAS ODD
```

```
0CB0- 8D 35 08 6900          STA FOUND.LEN   # TRAILING BLANKS
0CB3- 20 D4 0C 6910          JSR PRINT.Y.SPACES
              6920 *---LEFT JUSTIFY----------------
0CB6- 68      6930 .5        PLA             GET STRING LEN
0CB7- 20 C0 0C 6940          JSR PRT.STR     PRINT IT
0CBA- AC 35 08 6950          LDY FOUND.LEN   TRAILING SPACES
0CBD- 4C D4 0C 6960          JMP PRINT.Y.SPACES
              6970 *-------------------------------
              6980 PRT.STR
0CC0- 8D 36 08 6990          STA FOUND.CHAR    LEN OF STRING
0CC3- A0 FF    7000          LDY #$FF
0CC5- C8       7010 .1       INY
0CC6- CC 36 08 7020          CPY FOUND.CHAR
0CC9- B0 08    7030          BCS .2          DONE
0CCB- B1 F9    7040          LDA (P2),Y      GET CHAR
0CCD- 20 5C DB 7050          JSR AS.COUT     PRINT IT
0CD0- 4C C5 0C 7060          JMP .1
0CD3- 60       7070 .2       RTS
              7080 *-------------------------------
              7090 PRINT.Y.SPACES
0CD4- 98       7100          TYA             TEST COUNT
0CD5- F0 09    7110          BEQ .2          ...ZERO, EXIT NOW
0CD7- AD 37 08 7120          LDA FILL.CHAR
0CDA- 20 5C DB 7130 .1       JSR AS.COUT
0CDD- 88       7140          DEY
0CDE- D0 FA    7150          BNE .1
0CE0- 60       7160 .2       RTS
              7170 *-------------------------------
              7180 DP.VTAB
0CE1- 85 25    7190          STA MON.CV
0CE3- 4C 24 FC 7200          JMP MON.VTABZ
              7210 *-------------------------------
```

Symbol Table Source Maker.....Peter McInerney and Bruce Love

When developing a very large program in separately assembled stages, it is nice to be able to carry forward the information in the symbol table of one section into the equates section to later section. You might do this as a normal part of development or as response to a bug detected in an earlier stage which forces some re-assembly. We designed this utility program to take all the hard work out of the process of building an equate file from a symbol table.

After an assembly, BRUNning the following utility will cause whatever source is in memory to be replaced by a series of .EQ lines constructed from the current symbol table. All global labels are included, in numerical order. The generated source lines can be saved or merged in the usual fashion.

The plan of the program falls into three steps. First the existing symbol table is sorted into numeric order by the value of each symbol. Next a line corresponding to each symbol is constructed and merged into the source code. Finally the source lines are renumbered starting with 1000 using an increment of 10, and control is passed back to the S-C Macro Assembler.

We originally wrote our program based on Version 1.1 of the S-C Macro Assembler. Version 2.0 differs in that each symbol value uses four bytes rather than two, and the RENUMBER routine is in a different location. Bob Sander-Cederlof added some code to handle Version 2.0, and that version is listed here. All the changes that need to be made to use our utility with Version 1.1 are controlled by .DO-.ELSE-.FIN sets, so that you only have to change line 1030 to assemble the other version. Since the following listing was made with the CON listing option, the code between .ELSE and .FIN is shown as non-assembled lines; this allows you to type in both versions of the program.

After an assembly, the symbol table consists of 26 chains of symbols. A hash table of 26 pointers contains the beginning of each of the 26 chains. There is one chain for each letter of the alphabet, and symbols are assigned to a chain based on the first letter of the symbol name. Within each chain, the symbols are linked together in alphabetical order. The first two bytes of each symbol entry are a forward pointer to the next symbol in the chain, or $0000 if it is the end of the chain. If there is no chain for a particular letter, that pointer in the hash table will be $0000.

The value of the symbol is in the next two or four bytes (Version 1.1 or 2.0, respectively). The high byte of the value is first, the low byte last. The byte following the value contains the length of the symbol name in the lower six bits. The length will be a number between 1 and 32, or $01 and $20. Following the length byte are the characters of the name itself. Some other information is stored in the table, including various flags, local labels, and any macro definitions which were in your program; however, we are not concerned with these in our program.

The program begins by setting the output hook to point to our routine named MYCOUT. Any characters that are "printed" through the monitor's COUT routine will be routed to MYCOUT, at lines 2980-3070. MYCOUT merely stores the characters in successive positions of a buffer we put at $280. Lines 1350-1380 zap any source program still in memory, in preparation for adding the new .EQ lines.

Since every symbol carries a pointer, we decided to simply re-string them on a new chain in numeric order by value. Lines 1390-2040 build this new chain. Lines 1390-1490 and 1990-2040 step through each of the 26 alphabetical-order (A-O) chains. The numerical-order (N-O) chain is built with the pointer in ROOT pointing at the largest value, each symbol's pointer pointing at the next smallest value. When we find an A-O chain which is not empty, lines 1500-1980 chomp through the chain finding the right place in the N-O chain for each symbol.

Once the symbols are all strung on the N-O chain, lines 2050-2940 use the N-O chain to generate source lines for each symbol. Lines 2090-2100 check for the possibility of no symbols, just in case you are testing us.

Lines 2110-2210 pick up the value of the symbol (two or four bytes worth) and push it on the stack, low byte first. The loop actually pushes the byte following the value as well, because it saved a few program bytes to include it in the loop. Line 2220 pulls that byte back off.

Lines 2220-2280 pick up the characters of the symbol name and "print" them. Remember that the print hook points to MYCOUT, so that the characters are really placed in WBUF starting at WBUF+3. (The locations WBUF through WBUF+2 are reserved for the line length and line number.)

Lines 2290-2360 generate enough blanks to tab over to column 25. If the symbol is longer than 25 characters, only one blank is generated. All of the blanks are squeezed into a single compressed blank token ($80 + # of blanks). We put this into WBUF by calling MYCOUT1 to avoid the AND #$7F at the beginning of MYCOUT.

Lines 2370-2420 "print" the string of characters " .EQ $", which are stored in backwards order in line 3090.

Lines 2430-2610 "print" the value of the symbol in hexadecimal. Since the value may have up to three bytes of leading zeros, there is code here to suppress those bytes.

Lines 2620-2720 terminate the source line in WBUF with a $00 code, and store the line length in the first byte position. Now the line is ready to be added to the source code being built up.

Lines 2730-2790 make room for the new source line by lowering the pointer PRG.BEG, which points at the start of the source code. We are adding the source lines starting with the highest

value, which will be at the end of the source program, and working down to the lowest value at the beginning of the source program.

Lines 2800-2850 copy the line into the hole we just made. Note that we have not filled in a valid line number yet.

LInes 2860-2940 promote the ROOT pointer to the next symbol in the N-O chain. If there are no more symbols, line 2950 calls on the RENUMBER subroutine inside the S-C Macro Assembler to put real line numbers in each line. The point at which RENUMBER is entered is just after a series of three JSR's, all to the same address. The instruction we branch to is a "CPX #$06". We are pointing this out here just in case you have a version of the S-C Macro Assembler with a slightly different position for the RENUMBER subroutine. Of course, you could omit line 2950 and just remember to type "REN" after running our program.

Finally, line 2960 restores the output hook to the 40-column screen output. This will not be what you want if you are using an 80-column card. If you are doing that, we suggest saving the output hook way back at the beginning before stuffing MYCOUT into it, and then restoring the original value here. We didn't do it that way because we were trying every possible way to make this whole program fit in only one page.

One caveat remains. We did not include any test to see whether the source code being generated starts to overlap the end of the symbol table. If you have a gigantic symbol table, say over half of the available memory for source+symbols, you may run into this problem.

When you are using this program, be sure you save the source of whatever you assembled first. Our program replaces the source in memory with the .EQ source lines. Also, realize that the symbol table is essentially wiped out by running our program, because all the chain links are restructured for numerical order. You will have to re-assemble the original program to re-create the original symbol table. Of course, if you assemble the source lines we generate, you will re-create all the global labels of the original program.

We think you will find many uses for our program, beyond the ones which prompted us to write it. We are very proud that we managed to fit everything into a single page, but don't let that stop you from adding features to fit your own needs.

```
                1000 *SAVE S.SYMBOL SOURCEROR
                1020 *--------------------------------
01-             1030 VERSION    .EQ 1    0=1.1, 1=2.0
                1040 *--------------------------------
                1050     .DO VERSION      ...V 2.0
D64D-           1060 RENUMBER .EQ $D64D        V 2.0
                1070     .ELSE            ...V 1.1
                1080 RENUMBER .EQ $D7DA        V 1.1
                1090     .FIN
                1100 *--------------------------------
```

```
00-                 1110 PTR         .EQ $00,01
02-                 1120 A1          .EQ $02,03
04-                 1130 A2          .EQ $04,05
06-                 1140 ROOT        .EQ $06,07
08-                 1150 XSAVE       .EQ $8
36-                 1160 CSW         .EQ $36,37
                    1170 *-------------------------------
0132-               1180 HASH.TAB    .EQ $132
0280-               1190 WBUF        .EQ $280
                    1200 *-------------------------------
FDDA-               1210 PRBYTE      .EQ $FDDA
FDED-               1220 COUT        .EQ $FDED
FE93-               1230 SETVID      .EQ $FE93
                    1240 *-------------------------------
                    1250 *       PROGRAM POINTERS
                    1260 *-------------------------------
CA-                 1270 PRG.BEG     .EQ $CA,CB
4C-                 1280 PRG.END     .EQ $4C,4D
                    1290 *-------------------------------
                    1300 MAKE.SOURCE.FROM.SYMBOL.TABLE
0800- A9 F2         1310         LDA #MYCOUT        GRAB THE OUTPUT HOOK
0802- 85 36         1320         STA CSW
0804- A9 08         1330         LDA /MYCOUT
0806- 85 37         1340         STA CSW+1
0808- A5 4C         1350         LDA PRG.END        EMPTY THE PROGRAM AREA
080A- 85 CA         1360         STA PRG.BEG
080C- A5 4D         1370         LDA PRG.END+1
080E- 85 CB         1380         STA PRG.BEG+1
                    1390 *---SCAN THROUGH HASH TABLE------
0810- A2 00         1400         LDX #0
0812- 86 06         1410         STX ROOT           EMPTY NUMERIC-ORDER CHAIN
0814- 86 07         1420         STX ROOT+1
                    1430 *---GET START OF NEXT CHAIN------
0816- BD 33 01      1440 .1      LDA HASH.TAB+1,X
0819- F0 56         1450         BEQ .6                ...THIS CHAIN IS EMPTY
081B- 85 01         1460         STA PTR+1
081D- BD 32 01      1470         LDA HASH.TAB,X
0820- 85 00         1480         STA PTR
0822- 86 08         1490         STX XSAVE
                    1500 *---SEARCH FOR POSITION IN N-O CHAIN---
0824- A9 06         1510 .2      LDA #ROOT     START SEARCH FROM BEGINNING
0826- 85 02         1520         STA A1             OF NUMERIC-ORDER CHAIN
0828- A9 00         1530         LDA /ROOT
082A- 85 03         1540         STA A1+1
082C- A5 02         1550 .3      LDA A1        PROMOTE BOTH POINTERS
082E- 85 04         1560         STA A2             TO THE NUMERIC-ORDER CHAIN
0830- A5 03         1570         LDA A1+1
0832- 85 05         1580         STA A2+1
0834- A0 00         1590         LDY #0
0836- B1 02         1600         LDA (A1),Y
0838- AA            1610         TAX
0839- C8            1620         INY
083A- B1 02         1630         LDA (A1),Y
083C- 85 03         1640         STA A1+1
083E- 86 02         1650         STX A1
0840- F0 0D         1660         BEQ .5
                    1670 *---COMPARE A-O WITH N-O VALUE---
                    1680      .DO VERSION       ...V 2.0
0842- A2 03         1690         LDX #3         4-BYTE VALUES
                    1700      .ELSE             ...V 1.1
                    1710         LDX #1         2-BYTE VALUES
                    1720      .FIN
0844- 38            1730         SEC
0845- C8            1740 .4      INY
0846- B1 02         1750         LDA (A1),Y
0848- F1 00         1760         SBC (PTR),Y
084A- CA            1770         DEX
084B- 10 F8         1780         BPL .4
084D- B0 DD         1790         BCS .3         ...A-O VALUE < N-O VALUE
                    1800 *---INSERT A-O VALUE INTO N-O CHAIN---
084F- A0 00         1810 .5      LDY #0
0851- B1 00         1820         LDA (PTR),Y
0853- AA            1830         TAX
0854- A5 02         1840         LDA A1
0856- 91 00         1850         STA (PTR),Y
0858- A5 00         1860         LDA PTR
085A- 91 04         1870         STA (A2),Y
085C- C8            1880         INY
085D- B1 00         1890         LDA (PTR),Y
085F- 48            1900         PHA
```

```
0860- A5 03     1910          LDA A1+1
0862- 91 00     1920          STA (PTR),Y
0864- A5 01     1930          LDA PTR+1
0866- 91 04     1940          STA (A2),Y
0868- 86 00     1950          STX PTR
086A- 68        1960          PLA
086B- 85 01     1970          STA PTR+1
086D- D0 B5     1980          BNE .2          ...NOT END OF CHAIN YET
                1990 *---NEXT HASH CHAIN----------
086F- A6 08     2000          LDX XSAVE
0871- E8        2010 .6       INX
0872- E8        2020          INX
0873- E0 34     2030          CPX #2*26       26 HASH CHAINS
0875- 90 9F     2040          BCC .1          ...STILL ANOTHER CHAIN
                2050 *--------------------------------
                2060 *   RUN THROUGH NUMERIC-ORDER CHAIN
                2070 *   AND CREATE A SOURCE LINE FOR EACH SYMBOL.
                2080 *--------------------------------
0877- A5 07     2090          LDA ROOT+1      CHECK FOR NO CHAIN AT ALL
0879- F0 74     2100          BEQ .17
                2110      .DO VERSION         ...V 2.0
087B- A2 04     2120 .8       LDX #4
                2130      .ELSE               ...V 1.1
                2140 .8       LDX #2
                2150      .FIN
087D- A0 02     2160          LDY #2
087F- B1 06     2170 .9       LDA (ROOT),Y
0881- 48        2180          PHA
0882- C8        2190          INY
0883- CA        2200          DEX
0884- 10 F9     2210          BPL .9
0886- 68        2220          PLA
0887- 29 3F     2230          AND #$3F
0889- AA        2240          TAX
088A- B1 06     2250 .10      LDA (ROOT),Y
088C- 20 ED FD  2260          JSR COUT
088F- CA        2270          DEX
0890- D0 F8     2280          BNE .10
                2290 *---TAB TO .EQ COLUMN------------
0892- A9 81     2300          LDA #$81
0894- C0 19     2310          CPY #25
0896- B0 05     2320          BCS .11
0898- 98        2330          TYA
0899- 49 FF     2340          EOR #$FF
089B- 69 9A     2350          ADC #$9A
089D- 20 F4 08  2360 .11      JSR MYCOUT1
                2370 *---OUTPUT ".EQ $"---------------
08A0- A2 04     2380          LDX #4
08A2- BD F9 08  2390 .12      LDA STRING,X
08A5- 20 ED FD  2400          JSR COUT
08A8- CA        2410          DEX
08A9- 10 F7     2420          BPL .12
                2430 *---OUTPUT VALUE OF SYMBOL-------
                2440      .DO VERSION         ...V 2.0
08AB- A2 04     2450          LDX #4
08AD- 68        2460          PLA
08AE- D0 0A     2470          BNE .16         ...PRINT 32-BITS
08B0- CA        2480          DEX
08B1- 68        2490          PLA
08B2- D0 06     2500          BNE .16         ...PRINT 24-BITS
                2510      .ELSE               ...V 1.1
                2520          LDX #2
                2530      .FIN
08B4- CA        2540          DEX
08B5- 68        2550          PLA
08B6- D0 02     2560          BNE .16         ...PRINT 24-BITS
08B8- CA        2570          DEX
08B9- 68        2580 .13      PLA
08BA- 20 DA FD  2590 .16      JSR PRBYTE
08BD- CA        2600          DEX
08BE- D0 F9     2610          BNE .13
                2620 *---APPEND $00 BYTE--------------
08C0- 8A        2630          TXA             APPEND $00 BYTE
                2640      .DO VERSION         ...V 2.0
08C1- 99 7C 02  2650          STA WBUF-4,Y
08C4- 88        2660          DEY
08C5- 88        2670          DEY
                2680      .ELSE               ...V 1.1
                2690          STA WBUF-2,Y
                2700      .FIN
```

```
08C6- 88          2710          DEY
08C7- 8C 80 02    2720          STY WBUF      # BYTES IN LINE
                  2730 *---MAKE ROOM IN SOURCE AREA-----
08CA- A5 CA       2740          LDA PRG.BEG
08CC- 38          2750          SEC
08CD- ED 80 02    2760          SBC WBUF
08D0- 85 CA       2770          STA PRG.BEG
08D2- B0 02       2780          BCS .14
08D4- C6 CB       2790          DEC PRG.BEG+1
                  2800 *---COPY LINE INTO SOURCE AREA---
08D6- 88          2810 .14      DEY
08D7- B9 80 02    2820 .15      LDA WBUF,Y
08DA- 91 CA       2830          STA (PRG.BEG),Y
08DC- 88          2840          DEY
08DD- 10 F8       2850          BPL .15
                  2860 *---NEXT SYMBOL FROM CHAIN-------
08DF- C8          2870          INY           Y=0
08E0- B1 06       2880          LDA (ROOT),Y      FROM THE NUMERIC-ORDER CHAIN
08E2- AA          2890          TAX
08E3- C8          2900          INY
08E4- B1 06       2910          LDA (ROOT),Y
08E6- 85 07       2920          STA ROOT+1
08E8- 86 06       2930          STX ROOT
08EA- D0 8F       2940          BNE .8          ...NOT END OF CHAIN YET
08EC- 20 4D D6    2950          JSR RENUMBER    ...END, SO RENUMBER THE LINES
08EF- 4C 93 FE    2960 .17      JMP SETVID      RESTORE HOOK AND RETURN
                  2970 *-------------------------------
                  2980 MYCOUT
08F2- 29 7F       2990          AND #$7F
                  3000 MYCOUT1
08F4- C8          3010          INY
                  3020      .DO VERSION       ...V 2.0
08F5- 99 7B 02    3030          STA WBUF-5,Y
                  3040      .ELSE             ...V 1.1
                  3050          STA WBUF-3,Y
                  3060      .FIN
08F8- 60          3070          RTS
                  3080 *-------------------------------
08F9- 24 20 51
08FC- 45 2E       3090 STRING .AS "$ QE."
                  3100 *-------------------------------
                  3110 END
```

Short Single-Byte Hex-to-Decimal Printer...Bob Sander-Cederlof

Inside DOS there exists a subroutine whose purpose is to convert a single byte into a three digit decimal number, and print it out. It is called twice from within the CATALOG processor: to print the volume number, and to print the number of sectors in a file. It isn't very space or speed efficient, and has been picked apart in various articles in Nibble and elsewhere. The DOS routine is located at $AE42.

In any case, here is a shorter routine that does the same job. I also added a little test routine which exercises the subroutine by calling it for every possible value of a byte.

Lines 1200-1290 are the test routine. It is essentially equivalent to: FOR A = 0 TO 255 : PRINT X" "; : NEXT X.

Lines 1020-1160 are the conversion and print subroutine. It is written as a loop that runs the Y-register from 2 down to 0. Line 1030 starts Y=2, and lines 1140-1150 decrement and test Y, like BASIC's NEXT Y.

Another loop keeps subtracting a table entry from the value being converted until the remainder is smaller than the table entry. The table contains powers of ten. The first time through, 100 is subtracted as many times as possible. Each time, the X-register is incremented. Since line 1040 started X out as the ASCII code for zero, when the inner loop finishes X will have the ASCII code for the next decimal digit of the original value. Line 1120 calls the monitor COUT routine to print the digit.

The next time through the table value that gets subtracted is 10, and the third and last time through 1 gets subtracted. So you see that we first print the hundreds digit, then the tens digit, and finally the units digit.

The DOS version takes 40 bytes plus a three byte table, and mine takes 25 bytes plus a three byte table. It's probably not fair to compare 40 to 25 too unfavorably, because mine does use the X-register while the DOS version does not. The part of the CATALOG code that prints the number of sectors in a file requires that the X-register not be changed, so mine is not quite compatible as is. On the other hand, DOS goes to the trouble of saving the value to be printed in location $44, which is unnecessary, and also saves a value in $45 which is otherwise totally ignored. This foolishness takes place at $ADB9-$ADBF and $AE04-$AE0A.

```
                      1000 *SAVE S.PRINT 000-255
                      1010 *--------------------------------
                      1020 PRINT.000.255
0800- A0 02           1030         LDY #2
0802- A2 B0           1040 .1      LDX #"0"
0804- D9 19 08        1050 .2      CMP DECTBL,Y
0807- 90 06           1060         BCC .3          DIGIT FINISHED
0809- F9 19 08        1070         SBC DECTBL,Y
080C- E8              1080         INX
080D- D0 F5           1090         BNE .2          ...ALWAYS
080F- 48              1100 .3      PHA             SAVE REMAINDER
0810- 8A              1110         TXA
0811- 20 ED FD        1120         JSR $FDED
0814- 68              1130         PLA             GET REMAINDER
0815- 88              1140         DEY
0816- 10 EA           1150         BPL .1
0818- 60              1160         RTS
                      1170 *--------------------------------
0819- 01 0A 64        1180 DECTBL .DA #1,#10,#100
                      1190 *--------------------------------
081C- A9 00           1200 T       LDA #0
081E- 48              1210 .1      PHA             SAVE VALUE
081F- 20 00 08        1220         JSR PRINT.000.255
0822- A9 A0           1230         LDA #" "
0824- 20 ED FD        1240         JSR $FDED
0827- 68              1250         PLA             GET PREVIOUS VALUE
0828- 18              1260         CLC
0829- 69 01           1270         ADC #1          INCREMENT
082B- D0 F1           1280         BNE .1
082D- 60              1290         RTS
```


Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).